February | 2010

Thesis: Feature subset selection using support-vector machines by averaging over probabilistic genotype data

Francisco José Herrera Luque

Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Budapest University of Technology and Economics

Al presentar mi Proyecto de Fin de Carrera tengo la sensación de cerrar un ciclo, de superar una etapa, de coronar una cima. No ha sido sencillo, aquellos que me conocéis lo sabéis bien. En el ascenso ha habido momentos mejores y momentos peores, días de sol y días de lluvia, bellos senderos y paredes casi verticales. Miro hacia atrás y veo que en el camino me he dejado cosas, pero también me encontrado con otras muchas que no hubieran aparecido de haber cejado en el empeño. Miro hacia delante y veo ante mí nuevas puertas. Escarbo en los bolsillos y encuentro las llaves que fui recogiendo en mi andadura, creo que es momento de ver qué puertas abren.

Siento una gran gratitud por todos aquellos que a lo largo de la travesía me habéis servido de impulso de algún modo. Doy las gracias a mis padres, Paco y Reme, y a mi hermana Inma. Siempre me habéis brindado vuestro apoyo y sé qué siempre podré contar con vosotros. Es una de las pocas certezas que tengo en esta vida.

Doy a las gracias a mis amigos, a todos. A mis amigos del barrio, sois la conexión con mis orígenes. A mis amigos de la Universidad, tanto de clase como de estudio, ha sido una suerte teneros como compañeros durante el camino. A mis amigos de Erasmus, nunca olvidaré las cosas que compartimos. A mis amigos de aficiones y aventuras, que han sido muchas, por ayudarme a crecer en el resto de dimensiones de mi persona. Y en especial a mis mejores amigos. Ellos saben quienes son.

Doy las gracias a la gente a la que considero familia sin estrictamente serlo, ellos también saben quienes son. Doy gracias a los profesores que realmente lo son en el significado más estricto de la palabra. Me he cruzado con pocos, tanto en la Universidad como antes, pero me habéis dejado huella. Y doy las gracias al resto de personas de las que he aprendido algo, ya sea porque me lo han querido enseñar o porque de nuestro cruce de caminos supe sacar cosas buenas.

Aún sin haber dicho nombres sé que todos los aludidos podéis reconoceros en estas líneas. Seguramente me dejo personas en el tintero. Los errores son una parte esencial de la vida. Estoy seguro de que aquellos a los que no hago mención pero se saben apreciados por mí sabrán perdonármelo.

“La felicidad de tu vida depende de la calidad de tus pensamientos”.

Meditaciones de Marco Aurelio Antonino Augusto,

Emperador de Roma entre los años 161 y 180 d.C.

La persona que me hizo comprender esta frase me dio la lección más útil de mi vida.

1. Abstract

Despite the grand promises of the postgenomic era, such as personalized prevention, diagnosis, drugs, and treatments, the landscape of biomedicine looks more and more complex. The fullfillment of these promises for diseases significant in public health requires new approaches to induction for statistical and causal inferences from observations and interventions.

Within the biomedical world an important response to this challenge is the mapping and relatively cheap measuring of the genetic variations, such as single nucleotide polymorphisms (SNPs). The recent mapping of the genetic variations has opened a new dimension in the postgenomic research at all phenotypic levels, such as genomic, proteomic, and clinical, and it has sparked a series of Genetic Association Studies (GAS), based on the application of machine learning and data mining techniques.

To overcome such problems, different strategies are being investigated within the research community. The aim of this thesis work is to contribute to the progress in this field giving a step forward towards the solution. I have investigated the suitable machine learning and data mining algorithms for this task and the state of the art of the currently available implementations of them intended for biomedical research applications. As a result I have proposed a solution strategy, and chosen and extended the functionality of the Java-ML library, an open source machine learning library written in Java, implementing some missing algorithms and functionality that necessary for the proposed approach.

This thesis work is structured into three main blocks. Section 3 “An approach to the use of machine learning techniques with genotype data” addresses the faced problem and the proposed solution. It begins with the definition of some introductory GAS concepts and the description of the solution strategy and elaborates in subsequent subsections on the description of the theoretical underpinnings of the algorithms setting up the solution.

Specifically, the first subsection, “The feature selection problem in the bioinformatics domain”, justifies the necessity of reducing the dimensionality of data sets in order to allow for acceptable performance in the application of machine learning techniques to the broader field of bioinformatics implications and establishes a comparative taxonomy of the currently available techniques. In the second subsection, entitled “Feature selection using support-vector machines”, the idea behind support-vector machines classifiers and their application to feature subset selection is defined while the third subsection, “Ranking fusion as averaging technique: Markov chain based algorithms”, describes the ranking fusion algorithms which implementation has been chosen for the combination of the feature subsets obtained from different data sets.

Section 4 “Analysis of available tools for experimental design” analyses the available suitable tools for experimental design in GAS based on machine learning techniques. In this sense in the first subsection, “Advantages of high level languages for machine learning algorithms”, the convenience of using high level languages for the kind of applications we are working in is discussed. In the second subsection, “Machine learning algorithms implementations in Java”, the election of the Java language is justified followed by an analysis of the currently available implementations of machine learning algorithms in this language that are worthwhile to be considered for our purposes, namely WEKA, RapidMiner and Java-ML.

In Section 5 “Implemented extensions to the Java-ML library” a description of the functionalities that have been added to enable a framework suitable for the design of GAS experiments in order to test the proposed approach is provided. The “Missing values imputation: the dataset.tools package” subsection focuses on data sets handling functionalities while the “Averaging through ranking fusion: rankingfusion and rankingfusion.scoring package” subsection details the ranking fusion algorithms implementations. Finally the “How to use the code” subsection is a tutorial on how to use both the library and its extension for the development of applications.

In addition to these main blocks, a final section called “Future Work” reflects how the developed work can be used by GAS domain experts to evaluate the usefulness of the proposed technique.

2. Table of contents

3. An approach to the use of machine learning techniques with genotype data

The genotype of an organism is the set of inherited instructions it carries within its genetic code. A genetic locus or gene is a molecular unit of heredity of a living organism. A modern working definition of a gene [1] is "a locatable region of genomic sequence, corresponding to a unit of inheritance, which is associated with regulatory regions, transcribed regions, and or other functional sequence regions".

Not all organisms with the same genotype look or act the same way likewise not all organisms that look alike necessarily have the same genotype. In order to make clear the difference between an organism's heredity and what that heredity produces it is necessary to introduce the concept of phenotype. A phenotype is the composite of an organism's observable characteristics or traits, such as its morphology, development, biochemical or physiological properties, phenology, behaviour, and products of behaviour. Phenotypes result from the expression of an organism's genes as well as the influence of environmental factors and the interactions between the two.

By genetic association it is understood the occurrence, more often than can be readily explained by chance, of two or more traits in a population of individuals, of which at least one trait is known to be genetic. An allelomorph or allele is one of two or more forms of a gene (generally a group of genes). Sometimes, different alleles can result in different observable phenotypic traits, such as different pigmentation. However, many variations at the genetic level result in little or no observable variation. Studies of genetic association aim to test whether single-locus alleles or genotype frequencies (or more generally, multilocus haplotype frequencies) differ between two groups of individuals (usually diseased subjects and healthy controls). Such studies are based on the principle that genotypes can be compared "directly", i.e. with the sequences of the actual genomes.

A single-nucleotide polymorphism (SNP) is a DNA sequence variation occurring when a single nucleotide — A, T, C or G — in the genome (or other shared sequence) differs between members of a biological species or paired chromosomes in an individual. The recent mapping and relatively cheap measuring of the genetic variations, such as SNPs, has opened a new dimension in the postgenomic research at all phenotypic levels, such as genomic, proteomic and clinical, and it has sparked a series of Genetic Association Studies (GAS).

Machine learning techniques are increasingly popular in research fields like bio- and chemoinformatics [1], text and web mining, as well as many other areas of research and industry. However the application of such techniques in the bioinformatics field, and more specifically to the GAS is not straightforward because of the peculiarities of the available data sets.

Unfortunately, the application of machine learning and data mining techniques to SNP data is not straightforward. The performance of the available algorithms in the modeling task is dramatically diminished when applied to data sets containing SNP measurements because of their peculiarities. On one hand, such data sets usually include a small amount of samples for which a large number of features is defined. Traditional algorithms require exactly the opposite in order to perform well. On the other hand, the current techniques for SNPs measurement provide a relatively low precision, deriving in a high level of uncertainty in the data sets values, e.g. the genotyping is uncertain, or the missing value management provides uncertain imputation, or the haplotype reconstruction methods generate uncertain data.

Regarding my proposal, in order to reduce the input data sets dimensionality a possible strategy consists of selecting the subset of most relevant features, using the so called feature subset selection techniques. Such approach shows advantages compared to other alternatives, since it does not alter the original representation of the variables. As a consequence the original semantics of the variables are preserved, hence, offering the advantage of interpretability by a domain expert.

Despite this, the use of the inherently uncertain data sets containing genetic variations measurements still originates a strong variability in the results achieved by the feature selection techniques when applied to different data sets. It becomes necessary to combine the different feature subsets in such a way that the performance of the combination is optimized. A family of algorithms enabling this functionality is the family of ranking fusion techniques. In this work several variations of Markov chain based ranking fusion algorithms are investigated in addition to the Borda Count technique, which is the basic ranking fusion strategy and represents the reference against which performance measurements can be done.

Among the variety of feature subset selection techniques, the one based on the use of Support-vector Machines (SVMs) classifiers has been chosen for two reasons: first its performance over genetic variations measurements has still not been deeply investigated and second, it provides an ordering on the subset of selected features with respect to the target variable, what facilitates the combination of the results achieved over different data sets by means of ranking fusion.

Moreover, the existence of missing values in the data sets must also be faced. To this end a conservative approach has been followed, consisting of their estimation assuming the maximum level of uncertainty, that is, modelling the information generator with a uniform distribution which range is extracted from the available information.

The next three subsections elaborate on the description of the techniques that make up the proposed approach in order to provide the reader with understanding on their working principles and to justify their election. The first one justifies the necessity of reducing the dimensionality of data sets in order to allow for acceptable performance in the application of machine learning techniques to the broader field of bioinformatics and establishes a comparative taxonomy of the currently available techniques. In the second subsection, the idea behind support-vector machines classifiers and their application to feature subset selection is defined while the third subsection describes the ranking fusion algorithms which implementation has been chosen for the combination of the feature subset results obtained from different data sets.

3.1. The feature selection problem in the bioinformatics domain

As many pattern recognition techniques were originally not designed to cope with large amounts of irrelevant features, combining them with feature selection techniques has become a necessity in many applications [2]. The objectives of feature selection are manifold, the most important ones being: (a) to avoid overfitting and improve model performance, (b) to provide faster and more cost-effective models and (c) to gain a deeper insight into the underlying processes that generated the data.

However, the advantages of feature selection techniques come at a certain price, as the search for a subset of relevant features introduces an additional layer of complexity in the modeling task. Instead of just optimizing the parameters of the model for the full feature subset, we now need to find the

optimal model parameters for the optimal feature subset, as there is no guarantee that the optimal parameters for the full feature set are equally optimal for the optimal feature subset. As a result, the search in the model hypothesis space is augmented by another dimension: the one of finding the optimal subset of relevant features.

Feature selection techniques differ from each other in the way they incorporate this search in the added space of feature subsets in the model selection. In the context of classification, feature selection techniques can be organized into three categories, depending on how they combine the feature selection search with the construction of the classification model: filter methods, wrapper methods and embedded methods.

Filter techniques assess the relevance of features by looking only at the intrinsic properties of the data. In most cases a feature relevance score is calculated, and low-scoring features are removed. Afterwards, this subset of features is presented as input to the classification algorithm. Advantages of filter techniques are that they easily scale to very high-dimensional datasets, they are computationally simple and fast, and they are independent of the classification algorithm. As a result, feature selection needs to be performed only once, and then different classifiers can be evaluated.

A common disadvantage of filter methods is that they ignore the interaction with the classifier (the search in the feature subset space is separated from the search in the hypothesis space), and that most proposed techniques are univariate. This means that each feature is considered separately, thereby ignoring feature dependencies, which may lead to worse classification performance when compared to other types of feature selection techniques. In order to overcome the problem of ignoring feature dependencies, a number of multivariate filter techniques were introduced, aiming at the incorporation of feature dependencies to some degree.

Whereas filter techniques treat the problem of finding a good feature subset independently of the model selection step, wrapper methods embed the model hypothesis search within the feature subset search. In this setup, a search procedure in the space of possible feature subsets is defined, and various subsets of features are generated and evaluated. The evaluation of a specific subset of features is obtained by training and testing a specific classification model, rendering this approach tailored to a specific classification algorithm. To search the space of all feature subsets, a search algorithm is then ‘wrapped’ around the classification model.

However, as the space of feature subsets grows exponentially with the number of features, heuristic search methods are used to guide the search for an optimal subset. These search methods can be divided in two classes: deterministic and randomized search algorithms. Advantages of wrapper approaches include the interaction between feature subset search and model selection, and the ability to take into account feature dependencies. A common drawback of these techniques is that they have a higher risk of overfitting than filter techniques and are very computationally intensive, especially if building the classifier has a high computational cost.

In a third class of feature selection techniques, termed embedded techniques, the search for an optimal subset of features is built into the classifier construction, and can be seen as a search in the combined space of feature subsets and hypotheses. Just like wrapper approaches, embedded approaches are thus specific to a given learning algorithm. Embedded methods have the advantage that they include the interaction with the classification model, while at the same time being far less computationally intensive than wrapper methods. In the present work an embedded technique for feature selection based on the use of Support-Vector Machines classifiers has been investigated.

3.2. Feature selection using support-vector machines

In machine learning, a classification task usually involves training and testing data which consist of some data instances. Each instance in the training set contains one "target value" (class label) and several "attributes" (features). In mathematical terms, the training set can be defined as the set of instance-label pairs $\{(\bar{x}_i, y_i),\ i = 1, \ldots, l\}$ where $\bar{x}_i \in R^n$ and $y_i \in \{-1,1\}$. On the other hand, instances in the test set are defined only in terms of n-dimensional vectors of features. The goal of a classifier is, based on the information contained in the training set, to produce a model capable of predicting the target value of data instances in the testing set.

SVM (Support Vector Machine) is a useful technique for data classification. Despite its mathematical definition can be a bit involved, the idea laying behind is relatively simple. Given a training set of instance-label pairs, training vectors $\bar{x}_i$ are mapped into a higher (maybe infinite) dimensional space by a function φ. Then the SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. An intuitive view of the process is provided in Figure 1, that shows a toy bi-dimensional data set that is mapped into a three-dimensional space in which data can be linearly classified according their labels.

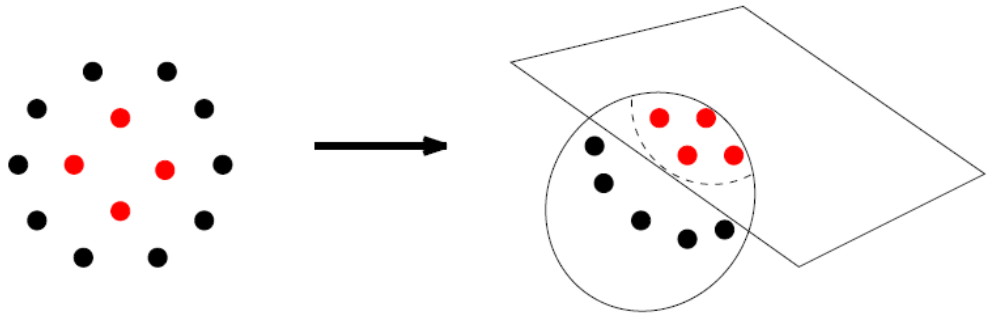

Figure 1. Graphical representation of the idea behind Support Vector Machines.

Mathematically speaking, the SVM [4] requires the solution of the following optimization problem:

$$\min_{\bar{w},b,\xi} \frac{1}{2}\bar{w}^T\bar{w} + C\sum_{i=1}^{l}\xi_i \text{ subject to } y_i(\bar{w}^T\varphi(\bar{x}_i) + b) \geq 1 - \xi_i,\ \xi_i \geq 0,$$

where $\bar{w}$ stands for the coefficients of the separating linear hyperplane, ξ_i is the distance error incurred by misclassified data instances (mapped data set might not be in general linearly separable), and $C > 0$ is the penalty parameter of the error term. Furthermore, $K(\bar{x}_i, \bar{x}_j) = \varphi(\bar{x}_i)^T\varphi(\bar{x}_j)$ is called the kernel function. A wide variety of kernels have being proposed by researchers. Figure 2 illustrates the relation between the parameters of the mathematical definition of the SVM and the separating hyperplane. Vectors $\bar{d}+$ and $\bar{d}-$ represent the shortest distances between training instances and the classifier border. Instances separated of the border by such distance are the so called "support vectors" and are the only ones determining the actual separating hyperplane.

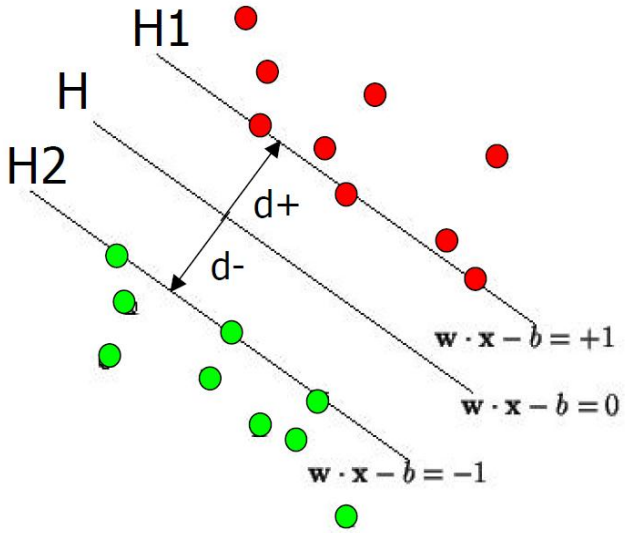

Figure 2. Training data and the corresponding SVM classifier hyperplane.

A ranking is a linear ordering of a set of items, features in our case. Different algorithms can be used to construct a ranking of features. One of them is defined by a recursive feature elimination procedure based on a support vector machine classifier [5]. The method is based upon finding those features which minimize bounds on the leave-one-out error. Starting with the full feature set, attributes can be ranked according to the weights they get in a SVM classifier. Subsequently, a percentage of the worst ranked features can be eliminated and the SVM can be retrained with the left-over attributes. The process can be repeated until only one feature is retained. The result is a feature ranking.

A ranking can be used as a means of feature subset selection choosing a number of the best ranked elements. In this work such algorithm has been used for feature subset selection though feature ranking, using for the SVM classifier a linear kernel function, defined as $K(\bar{x}_i, \bar{x}_j) = \bar{x}_i^T \bar{x}_j$.

3.3. Ranking fusion as averaging technique: Markov chain based algorithms

In order to combine the different feature subsets obtained from the application of the SVM based feature selection applied to different data sets we can take of advantage of the feature ranking provided by this technique. Thus the task of averaging the results consists of merging rankings [6] that provide different ordering of a common set of features.

Given a set of ranked lists or rankings, the task of ranking merging can be defined as the problem of combining these rankings in such a way that the performance of the combination is optimized. Among the multiple methods defined in the literature the focus is placed in this thesis work in those that are based in Markov chains. The trivial Borda Count method is also investigated since it represents the simplest approach to the ranking fusion problem and can be used as a reference for comparison.

Before defining each method it is necessary to introduce a brief mathematical framework. Let $U = \{1,2, \dots, |U|\}$ be a set of items, called *universe*. In our case, the universe contains the identifiers of the set of genes under study in the input data sets. A *rank list* (or simply ranking) τ with respect to U is

an ordering of the elements in U, *i.e.* $\tau = \{i \geq j \geq \cdots \geq k\}$ with $i, j, k \in U$ and ≥ some ordering relation on U. In other words, τ is a permutation of U, provided in our case by the SVM based ranking algorithm. With $\tau(i)$ it will be denoted the *position* of the gene i in the rank τ.

In the following we will consider a set of n rankings $R = \{\tau_1, \ldots, \tau_n\}$. Again in our case, the set of ranked elements (universe) is the same in every data set, but this is not the always the case in the general problem of ranking fusion. With $\hat{\tau}$ we will indicate the ranking (called fused ranking or fused rank list) which is the result of a rank fusion method applied to the rank lists in *R*. In order to fully specify $\hat{\tau}$, it will be sufficient to determine a score $s^{\hat{\tau}}(i)$ (called fused score), for each gene $i \in U$, as $\hat{\tau}$ will be ordered according to decreasing values of $s^{\hat{\tau}}(i)$.

A description on the chosen methods to calculate fused scores follows. The first one, called Borda Count, is the simplest so-called scoring ranking merging technique. Such family of methods relies on the association of an element score $\omega^{\tau}(i)$ for each of the ranked elements in each rank list as previous step to the determination of the fused score $s^{\hat{\tau}}(i)$.

In the Borda Count case, the element score is calculated for each ranked element based on its position in the rank, following the so-called rank normalization scoring policy $\omega^{\tau}(i) = 1 - \frac{\tau(i)-1}{|\tau|}$. This policy is monotone and increasing and assigns an evenly distributed weight with a difference between two subsequently ranked items of $1/|\tau|$. The top ranked item has a normalized rank element score of 1, while the bottom ranked item is assigned a score of $1/|\tau|$. The fused score is given by the expression $s^{\hat{\tau}}(i) = \sum_{\tau \in R} \omega^{\tau}(i)$, *i.e.* the fused score of each element is just the sum of the element's score in the different rankings.

An interesting approach to rank fusion is based on Markov chains. Markov chain methods do rely on comparison among the ranks only and neither considers scores. Let's introduce first the concept of Markov chain to facilitate the understanding of its application to the ranking fusion problem.

A (homogeneous) Markov chain for a system is a model specified by means of two elements: first, a set of states $S = \{1,2,\ldots,|S|\}$ in which the system can be and second, a matrix *M* of $|S| \times |S|$ dimensions, non-negative and stochastic (i.e. the sum of each row is 1) that defines the probabilities of state transitions. The system begins in some start state in *S* and at each step moves from one state to another state. The transition is guided by *M* as follows: at each step, if the system is at state *i*, it moves to state *j* with probability M_{ij}. An example three states Markov chain is illustrated in Figure 3 in a diagram form which corresponds to the transitions matrix $M = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$.

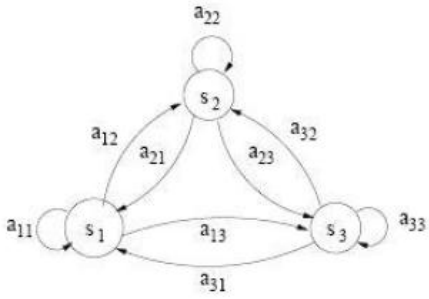

Figure 3. Three states Markov chain graphical representation.

If the current state is given as a probability distribution, the probability distribution of the next state is given by the product of the vector representing the current state distribution and *M*. In general, the start state of the system is chosen according to some distribution ***x*** (usually, the uniform distribution) on *S*. After *m* steps, the state of the system is distributed according to $\boldsymbol{x}M^m$.

Under some conditions (which we will not discuss here), irrespective of the start distribution ***x***, the system eventually reaches an unique fixed point where the state distribution does not change (few steps may suffice). This distribution is called stationary distribution. It can be shown that the stationary distribution is given by the principal left eigenvector ***y*** of *M*, i.e. $\boldsymbol{y}M = \boldsymbol{y}$. In practice, a simple power-iteration algorithm can quickly obtain a reasonable approximation of ***y***. The entries in ***y*** define a natural ordering on *S*. We call such an ordering the Markov chain ordering of *M*.

The application of Markov chains to the rank fusion problem is as follows. The set of states *S* corresponds to the list of all candidates to be ranked, *i.e.* the set of all items in $R = \{\tau_1, \dots, \tau_n\}$. The transition probabilities in *M* depend in some way on $\tau_1, \dots, \tau_n$, as proposed below, and $\hat{\tau}$ is then the Markov chain ordering on *M*. Note that the policies defined for the construction of the matrix *M* assume the general case in which not all the rankings contain the same elements.

MC1: if the current state is item *i*, then the next state is chosen uniformly from the multiset of all items *j* that were ranked higher than or equal to *i* by some ranking that ranked *i*, i.e. chose the next state uniformly from the multiset $Q_i^{C_1} = \cup_{k=1}^{n} \{j : \tau_k(j) \leq \tau_k(i)\}$;

MC2: if the current state is item *i*, then the next state is chosen by first picking a ranking τ uniformly from all the $\tau_1, \dots, \tau_n$ containing *i*, then picking an item *j* uniformly from the set $Q_{\tau,i}^{C_2} = \{j : \tau(j) \leq \tau(i)\}$;

MC3: if the current state is item *i*, then the next state is chosen as follows: first pick a ranking τ uniformly from all the $\tau_1, \dots, \tau_n$ containing *i*, then uniformly pick an item *j* that was ranked by τ. If $\tau(j) < \tau(i)$ then go to *j*, else stay in *i*;

MC4: if the current state is item *i*, then the next state is chosen as follows: first pick an item *j* uniformly from *S*. If $\tau(j) < \tau(i)$ for the majority of the lists $\tau \in R$ that ranked both *i* and *j*, then go to *j*, else stay in *i*.

4. Analysis of available tools for experimental design

In order to check whether the proposed strategy is feasible for the solution of the problems associated to the use of SNP measurements with machine learning techniques for the experimental design in GAS it is necessary to have the appropriate tools. A wide variety of machine learning algorithms implementations can be used, each of them enjoying different features. It is thus convenient to analyse the available options in order to choose the platform that better fits the specific needs of the problem under study. Such analysis is done in this section.

In the first subsection, the convenience of using high level languages for the kind of applications we are working in is discussed. In the second subsection, the election of the Java language is justified followed by an analysis of the currently available implementations of machine learning algorithms in this language that are worthwhile to be considered for our purposes, namely WEKA, RapidMiner and Java-ML.

4.1. Advantages of high level languages for machine learning algorithms

Machine learning algorithms implemented in some easy-to-use code have a much better chance of being widely adopted. There are mainly four important concerns associated with machine learning which stress programming languages on the ease-of-use vs. the speed frontier.

The first one is the rate at which data sources are growing, which seems to be outstripping the rate at which computational power does, so it becomes critical the ability to use the computational resources in an efficient way. Garbage collected languages (Java, Ocaml, Perl and Python) often have several issues here, which analysis follows.

Such languages often imply “boxed” floating point numbers, meaning every float is represented by a pointer to a float. Boxing can cause an order of magnitude slowdown because an extra non-localized memory reference is made, and accesses to main memory are many CPU cycles long.

In addition, usually considerably more memory than necessary is used. This has a variable effect: in some circumstances it results in no slowdown while in others it can cause a 4-order of magnitude slowdown. The red line in this sense is the avoidance of a situation in which a process runs out of physical memory and uses memory swapping.

Finally some of these languages are interpreted rather than executed. As a rule of thumb, interpreted languages are an order of magnitude slower than executed languages. Even when these languages are compiled, there are often issues with how well they are compiled since compiling to a modern desktop processor is a tricky business and only a few compilers do this well.

The second important concern is the ease of use of the language. It is a very subjective matter since from the developer point of view it is always easier to use the language that he or she is more familiar with. Familiarity isn’t just your familiarity, but also the familiarity of other people who might use the code. At one extreme, you can invent your own language. At the other extreme, you can use a language which many people are familiar with such as C or Java.

Nevertheless, significant differences exist between languages. Syntax is often overlooked, but it can make a huge difference in the ease of both learning to program and using the language. A good syntax allows for the study and the improvement of the algorithm rather than the program implementing it. Algorithmic improvements often yield the greatest speedups while optimizing. The syntax needs to be concise, so that it allow for viewing the entire algorithm, and simple, so that it can become second nature.

Library Support Languages vary dramatically in terms of library support, and having the right linear algebra/graphics/IO library can make a task dramatically easier. Built in Functionality Languages differ in terms of the primitives that are available. Which of these primitives are useful is a subject of significant debate. Some form of generic programming (templates, polymorphism, etc…) seems important. Functions as first class objects are often very convenient while programming. Built in lists and hash tables are also extremely useful primitives. One caveat here is that whenever making a speed optimization pass it is advisable to avoid these primitives. Support for modularity is also important. Objects (as in object oriented programming) are one example of this, but there are many others. The essential quantity here seems to be an interface creation mechanism.

The fourth and final concern is scalability and is where otherwise higher level languages often break down. A simple example of this is a language with file I/O built in that fails to perform correctly when the file has a size of 2^{31} or 2^{32} bytes. Special care must be taken in this sense.

4.2. Machine learning algorithms implementations in Java

Once the convenience of using a high level language for the implementation and use of machine learning algorithms has been justified comes the moment to choose one. Java is a general-purpose, concurrent, class-based, object-oriented language that has been specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another.

Compared to other high level languages, memory management is simplified through garbage collection, which help novices avoid things like memory leaks that can occur in a language like C (from which borrows much of its syntax) due to the accidental (mis)use of pointers.

In terms of speed, any well written Java program is a strong contender to an equivalent C/C++ program. Of course, C/C++ enjoy the benefit of ahead-of-time (AOT) compiling, where Java is compiled just in time (JIT), but the speed difference is probably insignificant.

Java has been around since the mid 1990's, and has been getting more and more popular since its creation. It has expanded from the desktop to servers with the Enterprise Edition (EE) and to mobile development with Mobile Edition (ME) and Android. It is nowadays one of the most popular programming languages in use, particularly for client-server web applications. For all these reasons, Java has been chosen in this thesis work.

Regarding the currently available implementations of machine learning algorithms in the Java, three initiatives are worthwhile to be mentioned: WEKA, RapidMiner and Java-ML. Their main features are free availability under the GNU General Public License, high portability (mainly because of its

implementation in Java and thus run on almost any modern computing platform), containing of a comprehensive collection of data pre-processing and modelling techniques, and ease of use by a novice due to the graphical user interfaces (WEKA and RapidMiner) and the simple programming interfaces (WEKA, RapidMiner and Java-ML). A brief description of each of them follows.

4.2.1. WEKA

WEKA (Waikato Environment for Knowledge Analysis) [7] is a machine learning algorithms workbench developed at the University of Waikato. It contains a collection of visualization tools and algorithms for data analysis and predictive modelling together with graphical user interfaces for easy access to this functionality.

The original non-Java version was a TCL/TK front-end to (mostly third-party) modelling algorithms implemented in other programming languages, plus data pre-processing utilities in C, and a Makefile-based system for running machine learning experiments. This original version was primarily designed as a tool for analyzing data from agricultural domains, but the more recent fully Java-based version (WEKA 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research.

It supports several standard data mining tasks, more specifically, data pre-processing, clustering, classification, regression, visualization, and feature selection. All supported techniques are predicated on the assumption that the data is available as a single flat file or relation, where each data point is described by a fixed number of attributes (normally, numeric or nominal attributes, but some other attribute types are also supported).

It provides access to SQL databases using Java Database Connectivity and can process the result returned by a database query. Despite it is not capable of multirelational data mining, there exists separate software for converting a collection of linked database tables into a single table that is suitable for processing using WEKA. Another important area that is currently not covered is sequence modeling.

WEKA's workbench is composed of panels. The main user interface is the Explorer, but essentially the same functionality can be accessed through the component-based Knowledge Flow interface and from the command line. There is also the Experimenter, which allows the systematic comparison of the predictive performance of machine learning algorithms on a collection of datasets. The Explorer interface has several panels that give access to the main components of the workbench.

The Preprocess panel provides facilities for importing data from a database, a CSV file, etc., and for pre-processing this data using a so-called filtering algorithm. These filters can be used to transform the data (e.g., turning numeric attributes into discrete ones) and make it possible to delete instances and attributes according to specific criteria.

The Classify panel enables the user to apply classification and regression algorithms to the resulting dataset, to estimate the accuracy of the resulting predictive model, and to visualize erroneous predictions, ROC curves, etc., or the model itself (if the model is amenable to visualization like, e.g., a decision tree). The Associate panel provides access to association rule learners that attempt to identify all important interrelationships between attributes in the data. The Cluster panel gives access to the clustering techniques, e.g., the simple k-means algorithm. There is also an implementation of the

expectation maximization algorithm for learning a mixture of normal distributions. The next panel, Select attributes provides algorithms for identifying the most predictive attributes in a dataset. The last panel, Visualize, shows a scatter plot matrix, where individual scatter plots can be selected and enlarged, and analyzed further using various selection operators.

4.2.2. RapidMiner

RapidMiner [8] (formerly YALE ,Yet Another Learning Environment) is an open source environment for machine learning and data mining experiments developed by the Artificial Intelligence Unit of the University of Dortmund since 2001.

In order to use it, it is necessary to interact with the Graphical User Interface (GUI) to design an analytical pipeline (the "operator tree" in RapidMiner parlance). The GUI generates an XML (eXtensible Markup Language) file that defines the analytical processes the user wishes to apply to the data. This file is then read by tool to run the analyses automatically. While these are running, the GUI can also be used to interactively control and inspect running processes. Other ways of using RapidMiner involve calling it from e.g., a Perl program. The Java Application Programming Interface (API) provides clear interfaces for applying operators individually (i.e., no need to create an operator tree), providing the ability to bypass the GUI and controlling analytical processes directly. Last, it is also possible to call individual functions directly from the command line.

RapidMiner is used for both research and real-world data mining tasks. It is well suited for analyzing data generated by high-throughput instruments, e.g., genotyping, proteomics, and mass spectrometry. It provides more than 500 operators for all main machine learning procedures, including input and output, and data preprocessing and visualization. In addition integrates learning schemes and attribute evaluators of the WEKA learning environment.

Although the core of this tool is open-source and is offered free of charge as a "Community Edition", there is also "Enterprise Edition", that is, according to the site, "Community Edition + More Features + Services + Guarantees". RapidMiner source is also offered under proprietary commercial license, to allow integration in closed-source solutions.

4.2.3. Java-ML

In contrast to the previously introduced data mining libraries, which provide a user-friendly interface and are geared towards interactive use, Java-ML [9], is a readily usable and extensible Application Programming Interface (API) oriented towards developers and research scientists that want to use machine learning in their own programs. To this end, interfaces are restricted to the essentials, and are relatively easy to understand, what facilitates a broad exploration of different models, the integration into new source code, and the library extension.

The library is built around two core interfaces: Dataset and Instance. These two interfaces have several implementations for different types of samples. In short every data sample is stored in an

Instance, and a set of Instances are grouped together in a Dataset. Each Instance can have a number of attributes or features that have real values and a class label.

Algorithms are functions that work on Datasets and Instances. Classification algorithms for example can be trained on a Dataset and can later be applied to classify Instances. Every machine learning algorithm implements one of the following interfaces: Clusterer, Classifier, FeatureScoring, FeatureRanking or FeatureSubsetSelection. Distance, correlation and similarity measures implement the interface DistanceMeasure. These distance measures can be used in many algorithms to modify their behavior. Cluster evaluation measures are defined by the ClusterEvaluation interface. Manipulation filters either implement InstanceFilter or DatasetFilter, depending on the level they work on.

Each of these interfaces provides one or two methods that are required to execute the algorithm on a particular data set. Several utility classes make it easy to load data from tab or comma separated files and from ARFF formatted files. All implementing classes for each of the interfaces are available from the API documentation that is available on the Java ML website.

The library provides several algorithms that have not been made available before in a bundled form. In particular, clustering algorithms and the accompanying cluster evaluation measures are extensively represented. This includes the adaptive quality-based clustering algorithm, density based methods, self-organizing maps (both as clustering and classification algorithm) and numerous other well-known clustering algorithms. A large number of distance, similarity and correlation measures are included. Feature selection algorithms include traditional algorithms like symmetrical uncertainty, gain ratio, RELIEF, stepwise addition/removal, as well as a number of more recent methods (SVMRFE and random forest attribute evaluation). Also the recently introduced concept of ensemble feature selection is incorporated in the library. It has also been implemented a fast and simple random tree algorithm to cope with high dimensional, sparse and ambiguous data. Finally, bridges for classification and clustering in WEKA and LIBSVM [10] are also provided.

5. Implemented extensions to the Java-ML library

Among the previously described libraries implementing machine learning algorithms in Java, Java-ML has been chosen for the development of the application implementing the proposed strategy. Several reasons have driven this election.

In contrast to the other two libraries, which provide a comfortable Graphical User Interface to facilitate the use of the algorithms over the input data, the interaction with Java-ML is through an API oriented towards developers and research scientists that want to use machine learning in their own programs. Thus, interfaces are restricted to the essentials, and are relatively easy to understand, what facilitates a broad exploration of different models, the integration into new source code, and the library extension.

Regarding its content, Java-ML has a different focus than the other libraries since it contains an extensive set of similarity based techniques, and offers state-of-the-art feature selection techniques. The feature selection techniques are well suited to deal with high dimensional domains, such as the ones often encountered in bioinformatics and biomedical applications. In the addition, the feature subsets ranking algorithm based on the use of Support-vector Machines already implemented in the library.

The library content has been extended in two directions. The first one permits the use of the application specific CSV files containing a data set of SNP measurements with missing values and a phenotypic target variable. The second extension consists of the implementation of the selected ranking fusion algorithms. A detailed description of such extensions follows.

5.1. Missing values imputation: the dataset.tools package

In order to implement the proposed solution it is necessary to wrap the available library functionality related with the representation of the data sets as internal data structures (`net.sf.javaml.core package`). The objective of this is twofold: on one hand to handle the existence of missing values in the SNP measurements with the proposed strategy, on the other hand, input files include data related to a phenotypic characteristic, which numerical format (float) cannot be handled by the library, which enforces the use of data with integer values. Such wrapper is implemented in a new class, `MyFileHandler`, inside the new `dataset.tools` package, which imitates the interface and structure of the library class `net.sf.javaml.tools.data.FileHandler`, performing the necessary ad-hoc processing to handle the input file format used in this work by making use of the library infrastructure.

Specifically, each record in the input file is represented as a `DenseInstance` library object, that is, a sample without missing values. Since records have missing fields this is done after missing values are assigned a random value from a uniform distribution in the range of values of the estimated field within the data source. A complete data set is represented by means of a `DefaultDataset` library object. Figure 4 depicts how the `MyFileHandler` class integrates with the library functionality.

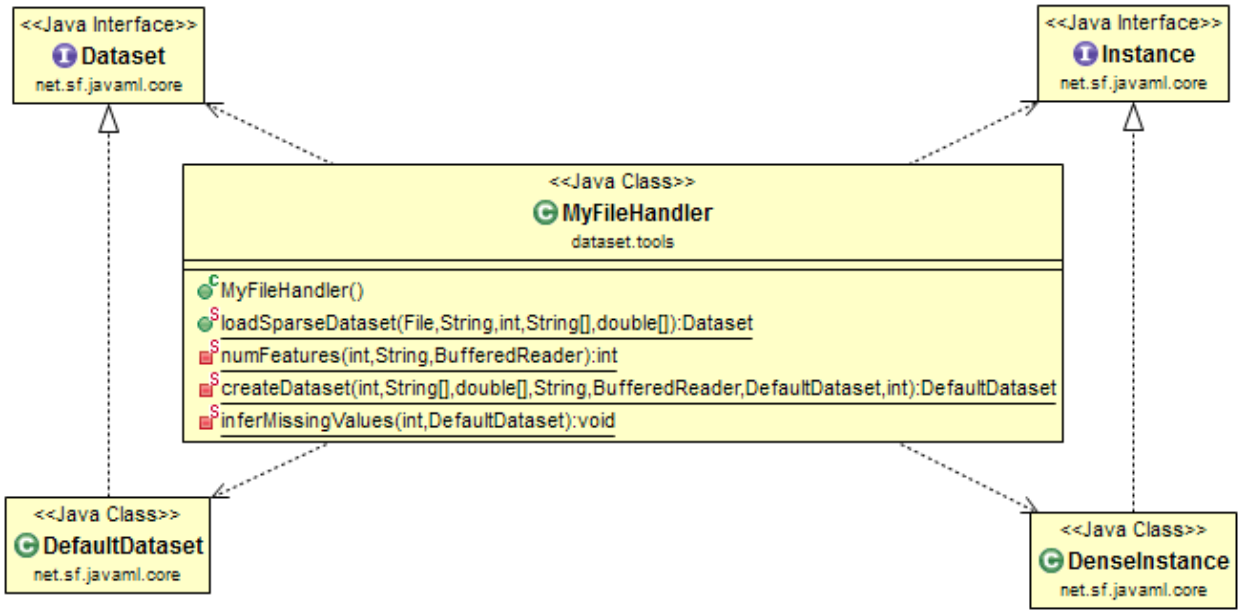

Figure 4. Library extension to handle the application specific data sets and its relation with library classes.

The target functionality of the class is implemented in the `loadSparseDataset` method. Since it is defined as public and static, there is no need for instantiating an object to invoke its execution. Five input arguments are needed, which description follows.

The first one is a comma-separated values (CSV) file containing SNP data of a set of individuals together with a target variable. A CSV file is a file format to store tabular data (numbers and text) in plain-text form. Plain text means that the file is a sequence of characters, with no data that has to be interpreted instead, as binary numbers. A CSV file consists of any number of records, separated by line breaks of some kind; each record consists of fields, separated by some other character or string, most commonly a literal comma or tab. The second argument of the function is the character used in the CSV file to separate fields.

For the case of the SNP data used in this application, records correspond to different patients and fields within each record define gene allele values for a set of selected genes and a specific phenotypic variable for each of the patients. Gene allele values are coded using integer numbers while the target variable has a real numerical value.

Here comes the problem. The data structures defined for the internal representation of records permits only integer numerical values, thus the target variable cannot be internally represented with its current numerical format. Phenotypic numerical measurements are typically used by medical specialists to classify individuals into a reduced number of categories for the concrete feature, i.e. the Body Mass Index measurements can be used to classify an individual according to its nutritional status into “underweight”, “normal weight” or “obese”, despite such classification can be refined into more categories.

For both reasons a simple and configurable classifier based on threshold comparison is implemented within the `loadSparseDataset` function, allowing for the definition of up to three

different categories for the phenotypic variable. The last three input arguments are related to this. The third one indicates the index of the CSV file field of the target variable while the fourth argument is an array of strings defining the categories' names and the fifth is an array of integers specifying the classification borders.

Figure 5 shows the execution flow of the `loadSparseDataset` method in a self-explanatory fashion. As it can be observed in such figure and also in Figure 4, it is factored into auxiliary private and static methods, so that the code is more readable and furthers improvements are easier.

First a buffered input stream is opened on the input data CSV file to enable its reading. After this, the first line of the input file is processed in order to determine the number of fields per record since it is required by the library infrastructure to internally represent each file record as an `Instance`. Such functionality is implemented in the `numFeatures` method. In the third step the whole file is processed by the `createDataset` method, which sets up the internal data structure representation of the information contained in the input file creating an Instance object per record and grouping them into a Dataset object. After this the input stream on the input file is closed since its information has already being loaded. The final step is the imputation of the missing values in the `Instances` corresponding to the blank fields in the input file. This is done by the `inferMissingValues` function generating a random value from a uniform distribution in the range of values of the estimated field within the data source.

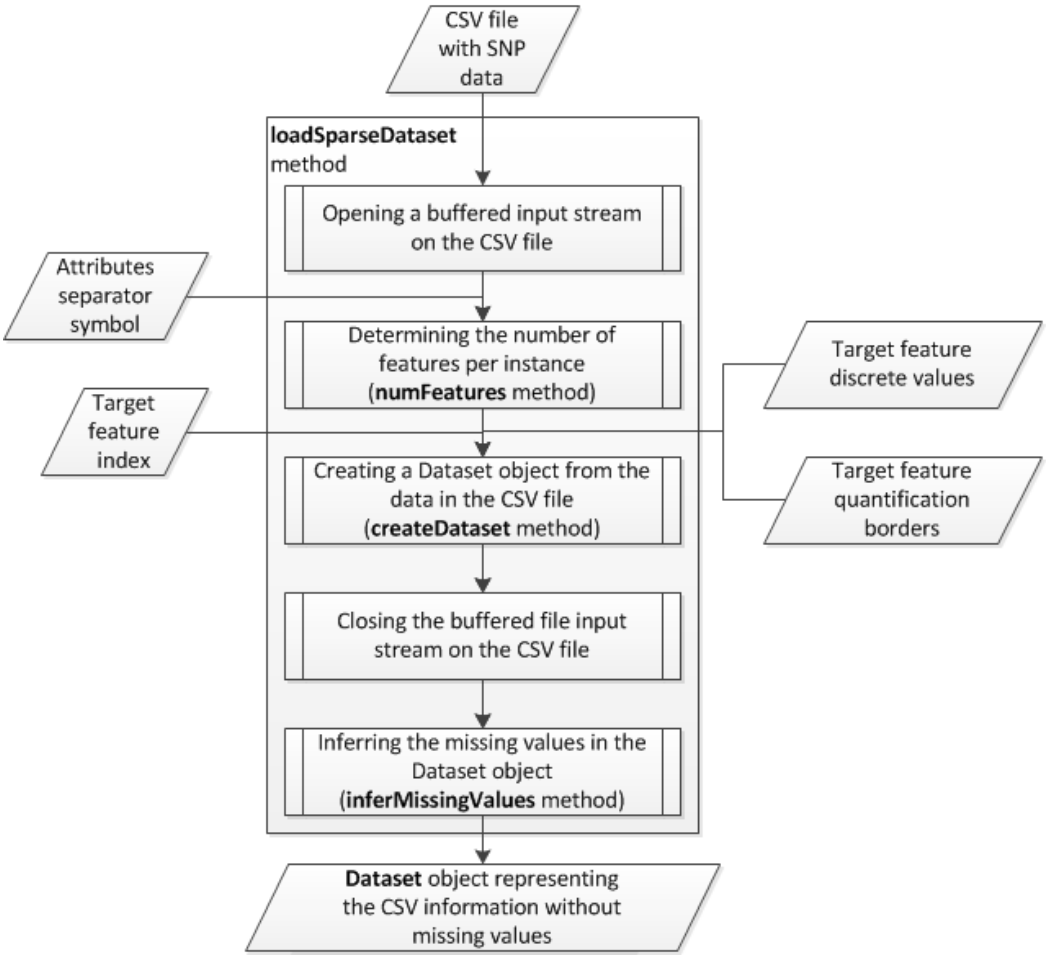

Figure 5. Execution flow of the loadSparseDataset method.

5.2. Averaging through ranking fusion: rankingfusion and rankingfusion.scoring packages

In relation to the averaging through ranking fusion strategy, the set of previously described algorithms for ranking fusion based on scoring have been implemented. The code has been structured following the philosophy of the library API into two different packages: `rankingfusion` and `rankingfusion.scoring`. The elements that have been defined and their relations are shown in Figure 6. In this case the relation with other library elements has not been represented to keep the illustration simple enough, but it mainly includes the `FeatureRanking` class belonging to the `net.sf.javaml.featureselection` package, which represents a set of ranked features.

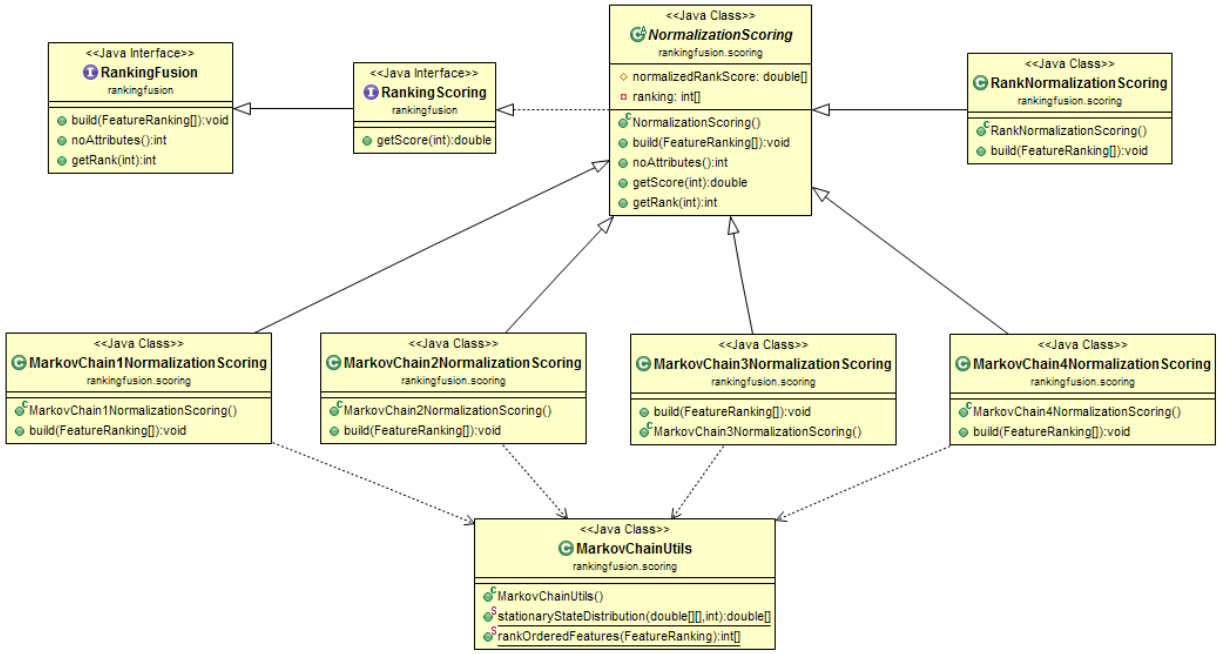

Figure 6. Library extension to implement ranking fusion algorithms.

The first package contains the definition of two interfaces, namely `RankingFusion`, which defines the methods that must implement the ranking fusion algorithms classes, and `RankingScoring`, which extends the `RankingFusion` and complements it with specific functionality for the scoring based ranking fusion algorithms. More precisely, the `RankingFusion` interface defines three methods: the `build` method, which implementation must perform the fusion of a set of rankings passed as input arguments, the `noAttributes` method, which invocation must return the number of elements composing the fused ranking, and `getRank`, through which the rank position of an input attribute ranked element is obtained. In the `RankingScoring` interface a method to retrieve the score associated to a specific ranked element in the algorithm scoring policy is defined. This structure imitates the philosophy of the `net.sf.javaml.featureselection` library package.

The `rankingfusion.scoring` package contains the classes implementing the `RankingScoring` interface according to the selected algorithms, namely `RankNormalizationScoring` (Borda Count method), `MarkovChain1NormalizationScoring` (implementing the Markov chain method previously defined as MC1), `MarkovChain2NormalizationScoring` (implementing the Markov chain method previously defined as MC2), `MarkovChain3NormalizationScoring` (implementing

the Markov chain method previously defined as MC3) and `MarkovChain4NormalizationScoring` (implementing the Markov chain method previously defined as MC4). Grouping these classes in this fashion, imitates the structure followed in the `net.sf.javaml.featureselection.ensemble`, `net.sf.javaml.featureselection.ranking`, `net.sf.javaml.featureselection.scoring` and `net.sf.javaml.featureselection.subset` library packages.

Since all these algorithms share some elements in their internal implementation, it becomes natural the introduction of an abstract (non instantiable) superclass in between the interface and actual algorithm implementations defining the common elements, so that the code is simplified. Such class is the `NormalizationScoring` class. It contains the two only necessary attributes for the execution of any ranking fusion algorithm, so that extending classes do not define them. Such attributes are a couple of internally accessible arrays that store the results of the ranking fusion algorithm execution through the `build` method invocation. One of them, `normalizedRankScore`, stores the score assigned to each ranked element, and the other, `ranking`, stores the rank position of each ranked element.

Regarding the methods, the part of the functionality of the `build` method common to every ranking fusion algorithm is implemented in the `NormalizationScoring` abstract superclass. It consists of the population of the ranking array based on the content of the `normalizedRankScore` array, which population has previously been performed by the corresponding ranking fusion algorithm in the corresponding subclass. In addition, all the other methods which implementation is enforced by the interface are defined in the superclass, since their functionality is reduced to the retrieval of information contained in the non-publicly accessible attributes.

As it can be deduced, the functionality implemented in the subclasses is the specific part of the each concrete ranking fusion algorithm, contained in the `build` method. Figure 7 shows the execution flow performed for the implementation of each of them. Details on specific algorithms are available in Section 3.3.

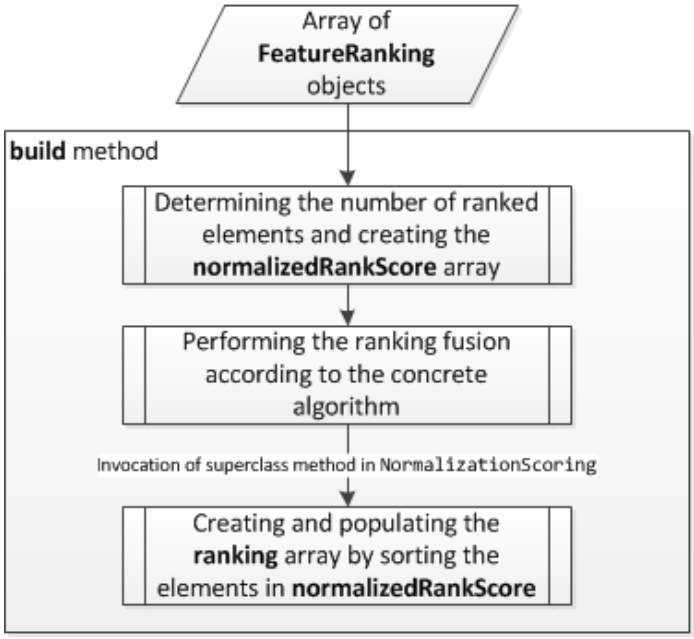

Figure 7. Execution flow of the ranking fusion algorithms build method.

The `RankNormalizationScoring` class implements the ranking fusion algorithm based in the Borda Count method. Since some calculations are common to all the Markov Chain based ranking fusion algorithms, they have been grouped in a separate class named `MarkovChainUtils`. Such calculations are the estimation of the stationary state distribution of a Markov chain using the algorithm based in exponentiating the transitions matrix (`stationaryStateDistribution` method), and the creation of an array containing the ranked elements ordered by its position in a given ranking (`rankOrderedFeatures` method).

5.3. How to use the code

This section intends to be a simple tutorial showing how to develop machine learning applications using both the Java-ML library and the library extensions implemented in this thesis work. It briefly describes how to install the available code and how to invoke the execution of the functionality implemented in the library extensions. Previous knowledge in Java language programming is assumed.

In order to enable the use of the implemented code it is necessary to install it before. The latest version of the Java-ML library can be downloaded from Sourceforge[1]. The downloaded file must be unpacked in a directory, preferable in the one that will be used to develop the new code. The library consists of a main jar-file (javaml-<version>.jar) and a number of support libraries that reside in the lib / directory. For ease of use, both the main Java-ML jar and the supporting libraries must be added to the system classpath. In case of use of an IDE like Eclipse or Netbeans, they also must be added to the build-path. After these steps the library is ready to be used. The same procedure must be followed with the jar files containing the library extensions implemented in this thesis work.

The use of the library code is documented in multiple tutorials available in the library URL. The implemented library extensions follow the library API style. A description on how to use them follows.

In order to load a CSV file containing SNP data sets, the `MyFileHandler.loadSparseDataset` method must be invoked passing the necessary arguments. Such arguments are: the path to the CSV file, the character used to separate consecutive fields in such file, the index of the column of the classifier's target variable in the file, an array containing the target variable category names for the target variable discretization process (as explained in section 6.1) and an array containing the target variable borders for such discretization process (also explained in section 6.1). A code example follows:

```
/* LOADING THE DATA IN THE CSV FILE */

//Path to CSV file in the file system
String path2CSVfile = "C:\Users\user1\data\data.csv";

//Symbol used to separate two fields in the CSV file
char separator = ',';

//Index of the column of the classifier's target variable in the CSV file
int targetVarPos = 85;

//Array of target variable names
String[] targetVarValues = {"low", "medium", "high"};
```

[1] http://sourceforge.net/projects/java-ml/files/

```
//Array of target variable borders for discretization
double[] targetVarBorders = {25.5, 29.5};

//Dataset object creation
Dataset dataset = MyFileHandler.loadSparseDataset(path2CSVfile, separator,
targetVarPos, targetVarValues, targetVarBorders);
```

The library extensions implementing ranking fusion algorithms need as an input an array of `FeatureRanking` objects that can be created based on Dataset objects using any of the available `FeatureRanking` available in the library. Of scientific interest for this thesis is the feature ranking algorithm based on support vector machines previously introduced, but any `FeatureRanking` object can be used with the ranking fusion algorithms. The code below shows the creation of a feature ranking from a `Dataset` object based on SVMs (note that `RecursiveFeatureEliminationSVM` implements the `FeatureRanking` interface):

```
/* Create a feature ranking algorithm */
RecursiveFeatureEliminationSVM svmrfeArff1 = new RecursiveFeatureEliminationSVM(0.2);
/* Apply the algorithm to the data set */
svmrfeArff1.build(dataset);
```

Assuming the previous creation of the `FeatureRanking` objects, the fusion of them using any of the implemented ranking fusion algorithms can be done as in the following example code. The process consists of instantiating the corresponding ranking fusion algorithm class and executing the `build` method passing as input parameter an array of FeatureRanking objects.

```
/* Array of rankings to be fused */
FeatureRanking [] rankings = new FeatureRanking[3];
rankings[0]=svmrfeArff1;
rankings[1]=svmrfeArff2;
rankings[2]=svmrfeArff3;

/* Create a fused ranking algorithm */
MarkovChain3NormalizationScoring mc3 = new MarkovChain3NormalizationScoring();
/* Apply the algorithm to the set of rankings to be fused */
mc3.build(rankings);
```

In order to analyse and compare the performance of the ranking fusion algorithms the `noAttributes`, `getScore` and `getRank` methods can be used. An example is use of these functions is the sample code below which prints the features ordered by their rank position after the fusion process has been performed.

```
/* Printing of the features ordered by their rank position */
System.out.println("Fused ranking mc3:");
for (int i = 0; i < mc3.noAttributes(); i++) {
      System.out.print(mc3.getRank(i)+" ");
}
System.out.println();
```

6. Future work

The first natural next step to complete the performed research is to contrast whether the proposed approach to improve the performance of the application of the machine learning techniques to the GAS is feasible or not. To do so it is necessary to carry out a series of GAS experimental design, developing a Java application that would use both the Java-ML library and the provided extensions. A sample experimental methodology can be found in [11].

Such experiments should target the detection of relevant genes in the prediction of a given phenotypic trait which genetic dependence is already known. Different design setups would be necessary in order to investigate the performance of the different elements composing the approach, namely, the imputation the missing the values, the convenience of the SVMs classifier based feature selection techniques and the determination of the best performing ranking fusion algorithms to handle the uncertainty associated to the SNP measurements. They all could be replicated at all phenotypic levels, such as genomic, proteomic and clinical.

Carrying out these experiments is beyond the scope of this thesis since the interpretation of the results requires a high level of expertise. In addition, gaining the rights over real genetic data is also a hard task with complex privacy concerns.

7. References

[1] Pearson H., "Genetics: what is a gene?", Nature 441 (7092): 398–401, 2006.

[2] Kanehisa, M. and Bork, P., “Bioinformatics in the post-sequence era”, Nature Genetics 33, 2003.

[3] Saeys, Y., *et al*., “A review of feature selection techniques in bioinformatics”, *Bioinformatics*, 23, 2507-2517, 2007.

[4] Cortes, Corinna and Vapnik, "*Support-Vector Networks*", Machine Learning, 20, 1995.

[5] S. Mukherjee, “Bioinformatics Applications and Feature Selection for SVMs”, class notes, 2001

[6] M. Elena Renda and Umberto Straccia, “Metasearch: Rank vs. Score Based Rank List Fusion Methods (without Training Data)”, I.E.I. - C.N.R. Technical Report, June 2002.

[7] http://www.cs.waikato.ac.nz/ml/weka/

[8] rapid-i.com/content/view/181/190/

[9] http://java-ml.sourceforge.net/

[10] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[11] Stein Aerts *et al.*, “Gene prioritization through genomic data fusion”, Nature Biotechnology, May 2006.

8. Resumen en español

8.1. Definición del problema y descripción del trabajo realizado

Se conoce como Aprendizaje Máquina a la rama de la Inteligencia Artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender. Se trata de un proceso de inducción del conocimiento mediante la creación de programas informáticos capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos. Su campo de actuación se solapa frecuentemente con la Estadística, ya que las dos disciplinas se basan en el análisis de datos.

El genotipo es la totalidad de la información genética que posee un organismo en particular, en forma de ADN. Junto con la variación ambiental que influye sobre el individuo, codifica su fenotipo. De otro modo, el genotipo puede definirse como el conjunto de genes de un organismo y el fenotipo como el conjunto de rasgos de un organismo.

Con un diseño experimental adecuado, es posible utilizar métodos estadísticos para correlacionar diferencias en los genotipos de las poblaciones con diferencias en sus fenotipos observados, en lo que se conoce como Estudios de Asociación Genética (Genetic Association Studies o GAS). Estos estudios de asociación se pueden utilizar para determinar los factores de riesgo asociados con una enfermedad. Pueden servir incluso para diferenciar entre poblaciones que pueden responder favorablemente o no a un tratamiento medicamentoso particular. En un estudio de asociación del genoma completo (Genome Wide Association study o GWA) se suele emplear como marcador el polimorfismo de nucleótido simple (en inglés Single Nucleotide Polymorphism, en lo sucesivo referido por sus siglas como SNP).

En este Proyecto de Fin de Carrera se aborda la problemática asociada a la utilización de técnicas de Aprendizaje Máquina en Estudios de Asociación Genética sobre conjuntos de datos con medidas de SNP y de una característica fenotípica concreta.

Dicha problemática radica en la baja efectividad de las técnicas de aprendizaje máquina empleadas sobre conjuntos de datos con medidas de SNP en comparación con otros tipos de datos. Ésta baja efectividad está originada por dos factores. Por una parte el elevado número de dimensiones (genes) presente en las muestras y el reducido número de muestras por conjunto de datos. Por otra parte por la relativamente alta incertidumbre asociada a los datos obtenidos en las medidas, en forma por ejemplo de errores de estimación ó ausencia de valores en algunas muestras para algunas de las dimensiones.

El trabajo realizado consiste en la proposición de una estrategia para mitigar dichos problemas, tras la realización de un estudio en profundidad, y el análisis y adaptación de las herramientas existentes para la implementación de dicha solución.

La estrategia propuesta combina diferentes técnicas, cada una de ellas con un propósito concreto. Para la reducción del número de dimensiones en los conjuntos de datos con medidas de SNP se propone el uso de técnicas de selección del subconjunto de variables más relevantes para la predicción de la variable objetivo, llamadas de Selección de Variables. De entre ellas se elige una técnica basada en el uso de Máquinas de Vector Soporte (Support-vector Machines o SVM).

Dos son las razones principales que justifican dicha elección, en primer lugar la aplicación de dicha técnica a la realización de GAS no se ha investigado suficientemente a dia de hoy, y en segundo lugar, como resultado de su aplicación se obtiene una ordenación según su relevancia para la predicción de la

variable objetivo del subconjunto de variables seleccionadas, lo que tal y como se verá a continuación, posibilita la estrategia implementada para afrontar la presencia de errores de estimación de la medida en los conjuntos de datos con medidas de SNP.

En relación a la incertidumbre asociada a las medidas de SNP en los conjuntos de datos se aborda por separado la presencia de errores en la estimación de las medidas y la ausencia de valores en algunas de las muestras para algunas de las dimensiones. Para mitigar el primer problema, se propone la combinación de los resultados obtenidos mediante las técnicas de selección del subconjunto de variables relevantes sobre conjuntos de datos diferentes, utilizando para ello técnicas de combinación de rankings aprovechando la ordenación por relevancia de las variables seleccionadas que proporciona la técnica basada en SVMs. Concretamente han sido seleccionadas para el estudio de su rendimiento en la realización de GAS un conjunto de técnicas basadas en cadenas de Markov, así como la técnica del "Recuento de Borda", por ser la estrategia más simple para la combinación de rankings y servir como referencia para la comparación.

Para tratar la ausencia de valores en algunas de las medidas, se ha adoptado un enfoque conservador en el que se asume el mayor grado de desconocimiento sobre el proceso generador de la información. Dicho enfoque consistente en la imputación de un valor aleatorio proveniente de una distribución de probabilidad uniforme definida sobre el dominio de la dimensión en cuestión dentro del conjunto de datos.

Para la implementación de esta estrategia se ha realizado en primer lugar un análisis de las implementaciones de acceso libre actualmente disponibles, en lenguajes de alto nivel y en concreto del lenguaje Java, de algoritmos de aprendizaje máquina. Como resultado se ha elegido la librería Java-ML, cuyas funcionalidades se han extendido para hacer posible la evaluación práctica de la solución planteada al problema tratado. Concretamente ha sido necesaria la implementación de las funcionalidades por una parte de representación con imputación de los valores ausentes de los conjuntos de datos con medidas de SNPs y de una característica fenotípica objetivo, y por otra parte de los algoritmos de combinación de rankings seleccionados para el estudio de su efectividad.

La continuación natural del trabajo realizado consiste en el diseño de experimentos de asociación genética que hagan uso de las herramientas desarrolladas y utilicen conjuntos de datos con información genotípica y fenotípica cuya correlación sea ya conocida. De este modo será posible evaluar el rendimiento de dichas herramientas y la validez de la solución propuesta para su utilización para determinar nuevas correlaciones entre las características genotípicas y fenotípicas.

8.2. Fundamentos teóricos de la solución propuesta

La mayoría de las técnicas de Aprendizaje Máquina existentes no han sido diseñadas para el manejo de conjuntos de datos con un gran número de variables incorrelacionadas con la variable objetivo. Tres son las ventajas principales que se obtienen mediante la utilización de técnicas de Selección de Variables como paso previo a la aplicación de las técnicas de Aprendizaje Máquina. En primer lugar se evita el sobre ajuste, mejorando así el rendimiento de los modelos, en segundo lugar se acelera el proceso de construcción de los modelos, disminuyendo así su coste, y por último se hace posible una mayor comprensión de los procesos subyacentes en la generación de los datos empleados. Como contrapartida la búsqueda del subconjunto de variables relevantes introduce un nivel adicional de complejidad en la tarea de construcción del modelo.

Las técnicas de Selección de Variables pueden clasificarse según la forma en que la búsqueda de las variables seleccionadas se integra con la búsqueda de los parámetros del modelo de los datos. De este modo surgen tres categorías según el grado de integración entre los procesos de selección de variables relevantes y de ajuste de los parámetros del modelo de datos: filtrado, envoltura y empotramiento. En las técnicas de filtrado ambos procesos son independientes y la selección de variables relevantes se efectúa en base a las propiedades intrínsecas de los datos, empleándose en la mayoría de los casos una política de puntuaciones. En las técnicas de envoltura, el conjunto de variables relevantes elegido se obtiene tras la evaluación del rendimiento de varios conjuntos de variables candidatos para la construcción del modelo de datos mediante una técnica concreta. Por su parte, en las técnicas empotramiento la búsqueda del conjunto de variables relevantes está totalmente integrada en el proceso de construcción del modelo.

En la estrategia propuesta en este estudio se utiliza una técnica de selección de variables relevantes del tercer tipo, basada en el entrenamiento de clasificadores del tipo Máquinas de Vector Soporte. La idea sobre la que se apoya este tipo de clasificadores es sencilla: dado un conjunto de datos etiquetados, se encuentra la transformación que aplicada sobre ellos permita su separación mediante una frontera de tipo lineal en un espacio con un mayor número de dimensiones. En lenguaje matemático el asunto consiste en la resolución del problema de optimización definido por:

$$\min_{\bar{w},b,\xi} \frac{1}{2}\bar{w}^T\bar{w} + C\sum_{i=1}^{l}\xi_i \text{ sujeto a } y_i(\bar{w}^T\varphi(\bar{x}_i) + b) \geq 1 - \xi_i, \xi_i \geq 0,$$

donde $\bar{w}$ representa los coeficientes del hiperplano lineal de frontera, ξ_i es la distancia de error asociada a los elementos erróneamente clasificados, y $C > 0$ es el parámetro de penalización del término de error. Al término $K(\bar{x}_i, \bar{x}_j) = \varphi(\bar{x}_i)^T\varphi(\bar{x}_j)$ se le llama función kernel. En la literatura pueden encontrarse diferentes definiciones para funciones de kernel. En este estudio se ha utilizado una función de kernel lineal, definida por $K(\bar{x}_i, \bar{x}_j) = \bar{x}_i^T\bar{x}_j$. Las distancias que separan la frontera del clasificador y los elementos más cercanos a ésta reciben el nombre de vectores soporte y dan nombre al clasificador, ya que son estos elementos los que condicionan la definición del hiperplano de separación.

Para la aplicación de este tipo de clasificador en la selección de subconjuntos de variables relevantes puede definirse un proceso recursivo de eliminación de variables basado en los coeficientes que se obtienen tras el entrenamiento de la SVM con un conjunto de datos etiquetados. En cada iteración se elimina un porcentaje de las variables peor clasificadas y se procede al rentrenamiento del clasificador con las variables restantes. El proceso termina cuando la dimensionalidad del conjunto de datos ha sido reducida a una sola variable, obteniéndose así una ordenación por relevancia para la

predicción de la variable objetivo del resto de las variables, es decir, un ranking.

La incertidumbre asociada a los datos de medidas de SNP utilizados en Estudios de Asociación Genética relativa a los errores en los valores estimados, tiene como consecuencia la no coincidencia en los resultados obtenidos a través de la aplicación de técnicas de Selección de Variables usando diferentes conjuntos de datos definidos sobre las observación de las mismas variables. Para consensuar un único subconjunto de variables relevantes, aprovechando la ordenación por relevancia que proporciona el algoritmo previamente expuesto, haremos uso de técnicas de combinación de rankings.

De entre ellas se ha elegido la familia de técnicas basada en el uso de cadenas de Markov por ser interesante conocer su rendimiento en los experimentos de Estudios de Asociación Genética. Adicionalmente se analizará la técnica conocida como Recuento de Borda por ser la manera más trivial de fusionar rankings, constituyendo una referencia cómoda con la que comparar los otros algoritmos.

Para la descripción de estas técnicas es necesario definir previamente un modelo matemático del problema. Llamaremos *universo* al conjunto de elementos $U = \{1,2,\dots,|U|\}$ formado por los identificadores de los genes presentes en los conjuntos de datos de entrada. Un *ranking* τ respecto de U es una ordenación de los elementos del conjunto, es decir $\tau = \{i \geq j \geq \cdots \geq k\}$ con $i,j,k \in U$ siendo ≥ una relación de orden definida sobre U. En otras palabras, τ es una permutación de U, obtenida en el nuestro caso mediante el uso del algoritmo de ordenación basado en SVMs. La *posición* de un gen i en el ranking τ se denota como $\tau(i)$.

En lo sucesivo consideraremos que tenemos un conjunto de *n* rankings $R = \{\tau_1,\dots,\tau_n\}$. Particularizando de nuevo a nuestro problema, el conjunto de elementos ordenados (universo) será el mismo en todos los rankings del conjunto. Este no es el caso en el problema general de la combinación de rankings. Mediante $\hat{\tau}$ nos referiremos al ranking que se obtiene como resultado de la combinación de los rankings en *R*, al que llamaremos *ranking fusionado*. Para su construcción será suficiente con determinar una puntuación $s^{\hat{\tau}}(i)$ (*puntuación fusionada*), para cada gen del universo, y ordenar estos en $\hat{\tau}$ según orden decreciente de $s^{\hat{\tau}}(i)$.

El método del "Recuento de Borda" es el más sencillo de la familia de métodos de combinación de rankings por puntuación. Este tipo de métodos se basan en asociar una puntuación a cada elemento y en cada ranking, que llamaremos $\omega^{\tau}(i)$, como paso previo al cálculo de la puntuación fusionada $s^{\hat{\tau}}(i)$. En el caso de Borda la puntuación se asigna en función de la posición del elemento en el ranking, siguiendo la llamada política de *puntuación normalizada*, definida por $\omega^{\tau}(i) = 1 - \frac{\tau(i)-1}{|\tau|}$. Se trata de una política de puntuación monótona creciente y de pesos uniformemente distribuidos, de modo que la diferencia entre dos elementos ordenados consecutivamente es de $1/|\tau|$. Al primer elemento del ranking se le asigna el valor 1, mientras que al último le corresponde $1/|\tau|$. La puntuación fusionada de cada elemento se obtiene mediante la suma de las puntuaciones asociadas en cada uno de los rankings, es decir $s^{\hat{\tau}}(i) = \sum_{\tau \epsilon R} \omega^{\tau}(i)$.

Una cadena de Markov (homogénea) se define como un conjunto de *estados* $S = \{1,2,\dots,|S|\}$ y una *matriz de transición de estados* ***M*** de dimensiones $|S| \times |S|$, no negativa y estocástica, que define las probabilidades de transición entre estados. Un sistema modelado mediante cadenas de Markov se encuentra en alguno de los estados del conjunto *S*. Si el sistema se encuentra en el estado *i* la probabilidad de transición al estado *j* viene dada por el elemento de la matriz de transición de estados $\boldsymbol{M_{ij}}$.

Si se modela la probabilidad de que un sistema se encuentre en cada uno de los estados mediante una distribución de probabilidad, dicha distribución evoluciona tras cada transición entre estados como el producto de la distribución actual, que llamaremos ***x***, y la matriz de transición de estados ***M***. Bajo ciertas condiciones (fuera del ámbito de esta discusión) el sistema converge a una distribución de probabilidad de sus estados estacionaria.

En la aplicación de las cadenas de Markov al problema de combinación de rankings se identifica en primer lugar el conjunto de estados *S* con la lista de elementos a ser ordenados, es decir el conjunto de elementos presentes en $R = \{\tau_1, \dots, \tau_n\}$. Para la construcción de la matriz de probabilidades *M* de transición se pueden definir diferentes algoritmos que emplean la información contenida en $\tau_1, \dots, \tau_n$. A continuación se proponen cuatro opciones que han sido implementadas en este trabajo. Cabe destacar que estos algoritmos no precisan que en todos los rankings aparezcan los mismos elementos.

MC1: si el estado actual es el elemento *i*, el siguiente estado se elige aleatoriamente de entre el subconjunto de elementos *j* cuya posición en los rankings es mejor o igual que la del elemento *i* en alguno de los rankings en los que aparece *i*, es decir, el siguiente estado se elige de forma equiprobable del subconjunto definido como $Q_i^{C_1} = \cup_{k=1}^{n} \{j: \tau_k(j) \leq \tau_k(i)\}$;

MC2: si el estado actual es el elemento *i*, para elegir el siguiente estado se elige en primer lugar de modo aleatorio uno de los rankings a fusionar τ de entre el conjunto de rankings $\tau_1, \dots, \tau_n$ en que aparece el elemento *i* y después se elige el estado de nuevo de forma aleatoria de entre los elementos de ese ranking con posición mejor ó igual, es decir, el siguiente estado se elige de forma equiprobable del subconjunto definido por $Q_{\tau,i}^{C_2} = \{j: \tau(j) \leq \tau(i)\}$;

MC3: si el estado actual es el elemento *i*, para elegir el siguiente estado se elige en primer lugar de modo aleatorio uno de los rankings a fusionar τ de entre el conjunto de rankings $\tau_1, \dots, \tau_n$ en que aparece el elemento *i* y se elige de forma aleatoria un elemento *j* presente en τ. Si la posición de *j* en el ranking es mejor que la de *i*, es decir si $\tau(j) < \tau(i)$, se elige *j* como estado siguiente, si no, se continúa en *i*;

MC4: si el estado actual es el elemento *i*, para elegir el siguiente estado se elige en primer lugar de modo aleatorio un elemento *j* de *S*. Si la posición de *j* en el ranking es mejor que la de *i*, es decir si $\tau(j) < \tau(i)$, en la mayoría de los rankings $\tau \in R$ en que aparecen tanto *i* como *j*, se elige *j* como estado siguiente, si no, se continúa en *i*;

8.3. Análisis de las implementaciones de técnicas de Aprendizaje Máquina

Como paso previo al diseño de experimentos que permitan contrastar la utilidad de la estrategia propuesta, es conveniente analizar y evaluar las herramientas que pueden ser utilizadas con este fin. La primera cuestión a abordar es el tipo de lenguaje de programación. La implementación de técnicas de Aprendizaje Máquina en lenguajes de alto de nivel resulta más ventajosa debido a la reducción de los costes de desarrollo de aplicaciones concretas pese a que su rendimiento sea menor en comparación con los lenguajes más cercanos a la máquina. No obstante, entre los lenguajes de alto nivel existen diversos factores a considerar de cara a mejorar su rendimiento en la aplicación que nos ocupa.

En primer lugar, los lenguajes con capacidades de recolección de basura (tales como Java, Ocaml, Perl y Python) hacen una gestión más eficiente de los recursos. No obstante, con este tipo de lenguajes hay que ser cuidadoso con el uso de variables numéricas de tipo “float”, ya que suelen ser referenciadas internamente mediante punteros, lo que implica cierta ralentización en la ejecución y el uso de más memoria. Además algunos de ellos son interpretados en lugar de compilados, lo que en general supone un enlentecimiento de alrededor de un orden de magnitud. A pesar de ello este factor es menos crítico de lo que podría pensarse ya que el rendimiento de muchos de los compiladores actuales deja bastante margen a la mejora.

El segundo factor a considerar es la sencillez de uso del lenguaje. Se trata de un aspecto bastante subjetivo que implica no solo las preferencias personales sino las de la comunidad de desarrolladores que puedan hacer uso del código que se desarrolle. Además, una sintaxis adecuada puede facilitar enormemente la mejora del rendimiento en un algoritmo.

Disponer de librerías que faciliten y den soporte al desarrollo de aplicaciones es otro elemento a tener en cuenta al elegir un lenguaje, sobre todo si éstas están relacionadas con el álgebra, los gráficos o las funciones de entrada y salida. Por último hay que prestar atención a las capacidades de escalabilidad del lenguaje ya que frecuentemente los lenguajes de alto nivel fallan al manejar archivos de entrada de gran tamaño.

En el presente estudio se ha elegido para el desarrollo de las herramientas necesarias para el contraste de la hipótesis de solución el lenguaje Java. Se trata de un lenguaje de alto nivel de propósito general, concurrente, basado en clases y orientado a objetos. Varias razones justifican su elección. La primera ventaja es que no necesita ser compilado en cada máquina en que se quiera ejecutar, sino que una vez compilado puede ser ejecutado en cualquier plataforma que disponga de máquina virtual de Java. Dispone de capacidades de recolección de basura, lo que evita incurrir en errores derivados del uso inadecuado de punteros que puede darse en lenguajes como C/C++, sin que haya grandes diferencias en cuanto a tiempo de ejecución comparándolo con ellos. Además se trata de un lenguaje de uso muy extendido, lo que sin duda constituye otra ventaja.

Existen varias librerías escritas en Java en las que se implementan algoritmos de Aprendizaje Máquina. De interés para este estudio son tres: WEKA, RapidMiner y Java-ML. De entre ellas se ha elegido la última para el desarrollo de las herramienta. Las dos primeras disponen de una interfaz gráfica de usuario que facilita su uso sobre archivos con datos, mientras que la interacción con Java-ML se realiza mediante una API concebida para su uso por investigadores interesados en desarrollar sus propias aplicaciones para la construcción de experimentos. Los interfaces que contiene se han definido de manera sencilla y manejan solamente los parámetros esenciales. Se facilita así su comprensión, la

integración dentro de código de desarrollo propio, la exploración de distintos y modelos e incluso la ampliación de las funcionalidades implementadas en la librería, como sucede en el caso que nos ocupa. Además la librería contiene la implementación de diferentes algoritmos de Selección de Variables, entre ellas el basado en el entrenamiento recursivo de SVMs previamente descrito.

8.4. Ampliaciones a la librería Java-ML

Ha sido necesario realizar una ampliación de las funcionalidades disponibles en la librería para posibilitar por una parte la representación de los conjuntos de datos con medidas de SNP con imputación de los valores ausentes y de una característica fenotípica objetivo, y por otra parte de los algoritmos de combinación de rankings.

Para la primera de las ampliaciones se ha desarrollado una clase de envoltura de la funcionalidad definida en la librería en el paquete `net.sf.javaml.core` para la representación en forma de estructuras de datos en memoria de la información contenida en los conjuntos de datos de entrada. Dicha clase, llamada `MyFileHandler`, se incluye en el paquete de nueva creación `dataset.tools`, e imita en su implementación el interfaz de uso y la estructura de la clase de la librería `net.sf.javaml.tools.data.FileHandler`. En la Figura 8 se muestra la estructura de dicha clase y su relación con otros elementos de la librería.

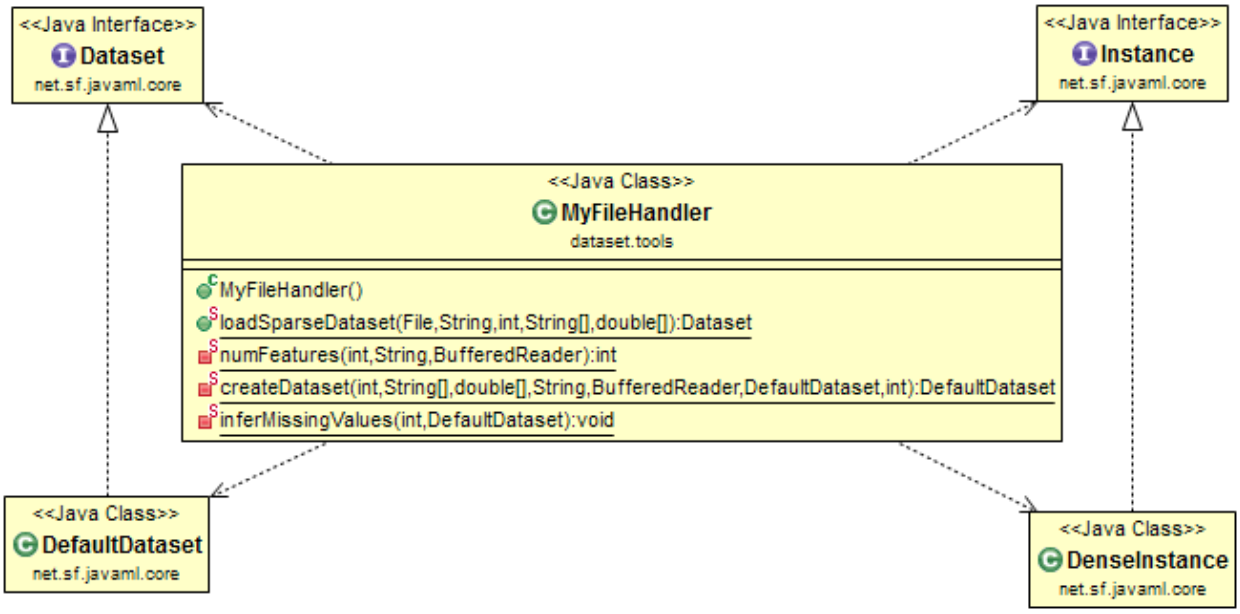

Figura 8. Amplición de la librería para el manejo de los archivos de datos de la aplicación que nos ocupa y su relación con las clases de la librería.

Para cargar en memoria los datos del fichero de entrada es necesario ejecutar el método estático `loadSparseDataset`, que requiere cinco parámetros de entrada. El primero es un archivo de tipo CSV (Comma Separated Values) con la información de medidas de SNP y de una variable fenotípica objetivo para un conjunto de individuos. El segundo es el carácter utilizado en dicho archivo para separar campos consecutivos. Antes de explicar los tres restantes es necesario describir las peculiaridades del formato de archivo CSV para el que está diseñada la aplicación que nos ocupa.

Los archivos CSV de entrada contendrán para cada una de los individuos (cada individuo se corresponde con una línea del archivo) el alelo que se identifica para una serie de genes, codificado

mediante un valor numérico entero, y el valor de la variable fenotípica objetivo expresada mediante un número real. Sucede que las estructuras de datos definidas en la librería para la representación de la información en los archivos de entrada impiden la utilización de variables objetivo para los clasificadores que sean de tipo real, es por esto que se hace necesario implementar un pre-clasificador basado en comparación con umbrales que discretice dichos valores. En la implementación se realiza dicha clasificación, hasta en tres categorías distintas.

Los tres argumentos de entrada restantes guardan relación con esta funcionalidad. El tercero especifica la posición (columna) dentro del archivo de entrada de la variable fenotípica objetivo entre el resto de variables, el cuarto es un array de cadenas de caracteres en el que se definen los nombres de las categorías del pre-clasificador y el quinto es un array con las fronteras de los intervalos de clasificación.

La segunda de las ampliaciones es la relativa a la implementación de los algoritmos de combinación de rankings previamente descritos. Para ello se ha estructurado el nuevo código en dos paquetes `rankingfusion` y `rankingfusion.scoring`. El primero de ellos contiene dos interfaces, `RankingFusion`, que define los métodos que deben implementar las clases en que se implementan los algoritmos de combinación de rankings, y `RankingScoring`, que extiende el interfaz anterior y lo complementa con los requisitos específicos para los algoritmos de fusión de rankings que emplean algún criterio de puntuación de los elementos de los rankings para la construcción del ranking fusionado. En la Figura 9 se muestran la clases implementadas en ambos paquetes y el modo en que se relacionan.

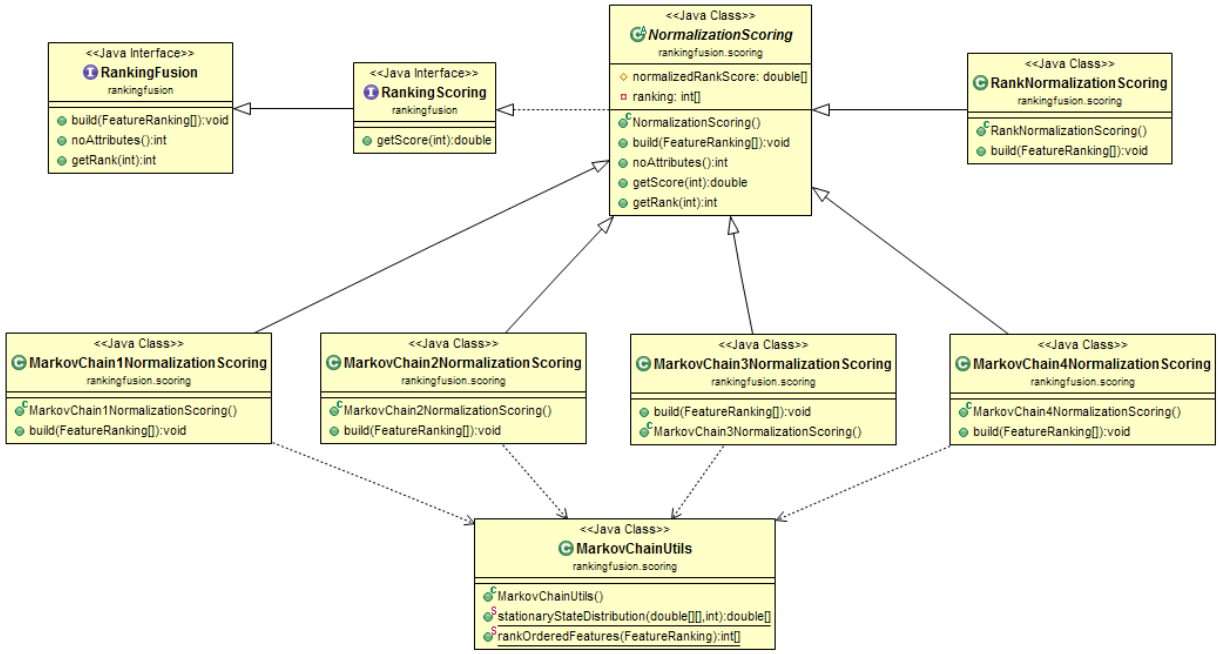

Figure 9. Ampliación de la librería para implementar los algoritmos de combinación de rankings.

El interfaz `RankingFusion` define tres métodos: `build`, cuya implementación debe generar el ranking fusionado a partir de un conjunto de rankings pasados como atributo de entrada en su invocación, `noAttributes`, que devuelve el número de elementos que contiene el ranking fusionado y `getRank`, que devuelve la posición en el ranking fusionado del atributo que se le pasa como parámetro en su invocación. El interfaz `RankingScoring` añade el método `getScore` para recuperar la puntuación que el algoritmo de combinación de rankings ha asignado internamente en su ejecución al atributo que se le pasa como parámetro en su invocación. Esta estructura extiende la librería imitando la

filosofía con que se define el API.

El segundo paquete contiene las clases que implementan el interfaz `RankScoring` para incluir la funcionalidad de los algoritmos seleccionados, que son: `RankNormalizationScoring` (algoritmo de "Recuento de Borda"), `MarkovChain1NormalizationScoring` (algoritmo basado en cadenas de Markov referenciado como MC1), `MarkovChain2NormalizationScoring` (algoritmo basado en cadenas de Markov referenciado como MC2), `MarkovChain3NormalizationScoring` (algoritmo basado en cadenas de Markov referenciado como MC3) and `MarkovChain4NormalizationScoring` (algoritmo basado en cadenas de Markov referenciado como MC4). Esta forma de empaquetamiento imita la estructura de los paquetes de la librería `net.sf.javaml.featureselection.ensemble`, `net.sf.javaml.featureselection.ranking`, `net.sf.javaml.featureselection.scoring` y `net.sf.javaml.featureselection.subset`.

Dado que todos ellos comparten ciertos segmentos de código en su implementación surge de manera natural la definición de una superclase no instanciable que contiene ese código, llamada `NormalizationScoring`. En ella se definen los dos únicos atributos necesarios para la ejecución de cualquiera de los algoritmos, dos arrays accesibles solamente desde el interior del objeto en los que se almacena la información generada como resultado de la ejecución. El primero, `normalizedRankScore`, almacena la puntuación asignada a cada elemento del ranking fusionado y el segundo, `ranking`, almacena la posición asignada a cada elemento en el mismo.

La parte de la funcionalidad del método `build` que es común a todos los algoritmos también se extrae y se implementa en la superclase. Dicha funcionalidad consiste en poblar el array `ranking` en función del contenido del array `normalizedRankScore`. El resto de métodos a cuya implementación obligan los interfaces también se definen en `NormalizationScoring`, ya que su funcionalidad consiste en proporcionar acceso a la información contenida en los atributos. Por tanto en las subclases se implementa solamente la parte del método `build` relativa a las peculiaridades de cada uno de los algoritmos de combinación de rankings.

Finalmente, dado que algunos cálculos son comunes a todos los algoritmos basados en cadenas de Markov se han agrupado en una clase separada llamada `MarkovChainUtils`. Estos cálculos son concretamente la estimación de la distribución estacionaria de estados de una cadena de Markov basada en exponenciación de la matriz de transiciones (método `stationaryStateDistribution`) y la creación del array que contiene los elementos ordenados en el ranking (método `rankOrderedFeatures`).

Para el desarrollo de aplicaciones es necesario en primer lugar descargar[2] e instalar la librería. El archivo descargado debe ser descomprimido preferentemente en el mismo directorio en el que vaya a ubicarse el código que se desarrolle. La librería se compone de un archivo jar principal (javaml-<version>.jar) y una serie de librerías de apoyo incluidas en el directorio lib/. Para facilitar su uso es conveniente añadir al classpath del sistema tanto el archivo principal como las librerías de apoyo. Si se usa algún IDE como Eclipse o Netbeans, también habría que añadirlos al build-path. Para añadir las extensiones a la librería desarrolladas en el este PFC habría que seguir el mismo procedimiento con los archivos jar que las contienen.

En la URL de la librería están disponibles abundantes tutoriales sobre su utilización. Las extensiones implementadas respetan la filosofía de la librería. El siguiente fragmento de código muestra cómo

[2] http://java-ml.sourceforge.net/

utilizar el método `MyFileHandler.loadSparseDataset` para la carga del conjunto de datos contenido en un archivo CSV con las características previamente definidas.

```
/* LOADING THE DATA IN THE CSV FILE */

//Path to CSV file in the file system
String path2CSVfile = "C:\Users\user1\data\data.csv";

//Symbol used to separate two fields in the CSV file
char separator = ',';

//Index of the column of the classifier's target variable in the CSV file
int targetVarPos = 85;

//Array of target variable names
String[] targetVarValues = {"low", "medium", "high"};

//Array of target variable borders for discretization
double[] targetVarBorders = {25.5, 29.5};

//Dataset object creation
Dataset dataset = MyFileHandler.loadSparseDataset(path2CSVfile, separator,
targetVarPos, targetVarValues, targetVarBorders);
```

Las extensiones que implementan algoritmos de combinación de rankings requieren como parámetro de entrada un array de rankings, definidos mediante objetos que implementan el interfaz `FeatureRanking`, que se pueden crear en base objetos de tipo `Dataset`. De interés para el problema que nos ocupa es el algoritmo de ordenación de características por su relevancia para la clasificación mediante Máquinas de Vector Soporte, disponible en la librería en la clase `RecursiveFeatureEliminationSVM`, aunque cualquier otro puede ser utilizado. El código de ejemplo que sigue muestra cómo crear un ranking con dicho algoritmo en base a un conjunto de datos ya cargado en memoria.

```
/* Create a feature ranking algorithm */
RecursiveFeatureEliminationSVM svmrfeArff1 = new RecursiveFeatureEliminationSVM(0.2);
/* Apply the algorithm to the data set */
svmrfeArff1.build(dataset);
```

Asumiendo la existencia previa de lo objetos de tipo `FeatureRanking`, para combinarlos empleando cualquiera de los algoritmos implementados en la extensión a la librería tan solo es necesario instanciar la clase del algoritmo correspondiente y ejecutar el método `build` pasando como parámetro de entrada un array que contenga los rankings, tal y como se ejemplifica en el siguiente código:

```
/* Array of rankings to be fused */
FeatureRanking [] rankings = new FeatureRanking[3];
rankings[0]=svmrfeArff1;
rankings[1]=svmrfeArff2;
rankings[2]=svmrfeArff3;

/* Create a fused ranking algorithm */
MarkovChain3NormalizationScoring mc3 = new MarkovChain3NormalizationScoring();
/* Apply the algorithm to the set of rankings to be fused */
mc3.build(rankings);
```

Finalmente si se necesita analizar o comparar el rendimiento de los algoritmos de combinación de rankings se pueden utilizar los métodos `noAttributes`, `getScore` y `getRank`. Un ejemplo de uso se muestra en el siguiente código, que sirve para imprimir los identificadores de los genes en el orden de relevancia para la predicción de la variable objetivo en que han quedado tras el proceso de combinación rankings.

```
/* Printing of the features ordered by their rank position */
System.out.println("Fused ranking mc3:");
for (int i = 0; i < mc3.noAttributes(); i++) {
      System.out.print(mc3.getRank(i)+" ");
}
System.out.println();
```